Estudi comparatiu del comportament d'unitats funcionals davant la injecció d'errors

FACULTAT D'INFORMÀTICA DE BARCELONA (FIB)
UNIVERSITAT POLITÈCNICA DE CATALUNYA (UPC) – BarcelonaTech

Autor: Marc Portavella Boixader
Director: Ramon Canal Corretger

Grau en Enginyeria Informàtica
Especialitat Enginyeria dels computadors

21 d'octubre de 2015

Agraïments

M'agradaria agrair al meu director, Ramon Canal, per acceptar portar el projecte així com l'ajuda i el suport que m'ha donat tant en el plantejament del mateix, com en les fases de desenvolupament.

També vull donar gràcies pel suport i els consells rebuts per part d'alguns companys de carrera, ja que aquella idea per solucionar algun dels problemes que han sorgit, o aquell suggeriment per tal d'estructurar els continguts d'aquest document, han fet la feina més fàcil.

Per últim agrair a la família el recolzament incondicional que m'ha donat, ells m'han donat l'oportunitat de cursar uns bons estudis i sempre han cregut en mi.

Resum

En aquest document es mostra i s'explica la importància de tenir en compte i estudiar l'efecte que tenen els errors transitoris en el *hardware* dels processadors.

S'ha simulat i estudiat la injecció d'errors en diverses unitats funcionals amb l'objectiu de comparar i analitzar quina d'elles té més tolerància davant d'aquests errors i per tant a la llarga és més fiable.

L'estudi realitzat compara vuit implementacions diferents de sumadors tan analíticament com empíricament amb un nombre estadísticament significatiu de simulacions.

Resumen

En este documento se muestra y explica la importancia de tener en cuenta y estudiar los efectos que tienen los errores transitorios en el hardware de los procesadores.

Se ha simulado y estudiado la inyección de errores en diferentes unidades funcionales con el fin de comprar y analizar cual de elles es mas tolerante a estos errores y por lo tanto mas fiable a largo plazo.

El estudio realizado compara ocho implementaciones diferentes de sumadores a nivel teórico y también empírico con un numero estadísticamente significativo de simulaciones.

Abstract

This paper shows and explains the importance of taking into account and have studied the effect of transient errors in hardware processors.

It has been studied and simulated the injection of errors in various functional units in order to compare and analyze which one has more tolerance for these errors and therefore ultimately is more reliable.

The study compares eight implementations of adders both analytically and empirically using an statistically significant number of simulations.

Índex

Il·lustracions

Taules

1. Introducció

1.1. Formulació del problema

Avui en dia, a mesura que avança el desenvolupament tecnològic en l'àmbit dels processadors, ens trobem amb el problema que com més petits es fabriquen els transistors amb l'objectiu d'augmentar-ne el nombre donat el mateix espai, més susceptibles són a veure's afectats per l'impacte de partícules, provocant un funcionament incorrecte d'algun dels seus circuits.

Aquest projecte vol treballar sobre la importància que tenen els errors transitoris (com l'impacte de partícules) sobre els dispositius electrònics segons la seva implementació. Així doncs, aquest projecte consistirà en fer un estudi comparatiu entre sumadors, que es farà amb la intenció de veure com respon cada tipus d'implementació davant la injecció d'errors. Amb això es vol simular els possibles errors que poden sorgir a les unitats en haver-hi el canvi del valor d'un bit en un punt del circuit a causa de l'impacte de partícules, i així poder comprovar quina d'elles és més robusta davant aquest tipus de fallades.

La recerca en aquest àmbit és important ja que el problema plantejat cada cop és més important. És interessant sobretot per en un futur millorar i poder assegurar una bona fiabilitat dels diversos productes que utilitzen processadors, que avui en dia són molts.

Una altra part de projecte serà veure i analitzar dos mètodes d'implementació dels circuits tenint en compte la posterior injecció d'errors, ja que no és igual implementar per experimentar sobre *FPGA*(*Field Programmable Gate Array*) o fer-ho mitjançant simulació.

1.2. Motivació

El motiu principal que ha portat a l'elecció d'aquest projecte ha sigut la continuïtat que dona a algunes assignatures de l'especialitat cursada, com ara el projecte de l'especialitat(PEC) i l'assignatura VLSI.

S'ha volgut aprofundir en la lògia de circuits, així com a en la seva implementació. Això s'ha aconseguit treballant amb un llenguatge nou de descripció i amb dissenys nous no vistos amb anterioritat.

Aquest projecte també introdueix el conjunt que és la injecció d'errors i les proves per validar i avaluar *hardware.* Àmbit important i sobre el qual no s'havia treballat abans.

1.3. Objectius

Un cop explicat el problema i vist com es relaciona amb el projecte, els objectius del projectes són:

- Estudiar l'efecte i la importància que té l'impacte de partícules en els processadors, concretament sobre algunes unitats funcionals d'aquests.

- Comprovar la robustesa davant els errors que provoquen els impactes segons el tipus d'implementació de la unitat funcional. Inicialment s'estudiaran les següents implementacions de sumadors: *Ripple carry*, *Carry select*, *Carry skip*.

- Comprovar la robustesa de cada implementació segons les portes que s'han utilitzat per implementar el circuit, en el cas dels sumadors:
  *AND*s i *XOR*s, *NAND*s i per últim *NOR*s.

- Arribar a determinar la implementació optima i més fiable davant l'impacte de partícules d'entre les estudiades.

2. Context

Així doncs aquest projecte pretén aprofundir en l'estudi sobre l'impacte de partícules i el seu efecte sobre els dispositius d'avui en dia.
Aquesta temàtica va guanyant importància a mesura que el desenvolupament tecnològic dels processadors avança [1] [2]. De mica en mica augmenta l'interès en l'estudi d'aquest camp. Una recerca solida en aquest camp, és molt interessant per part dels fabricants per augmentar la fiabilitat dels seus aparells i fidelitzar el client final.

La investigació en aquest camp és complicada, ja que hi ha molts factors responsables en l'impacte de partícules, siguin meteorològics, geogràfics o purament tecnològics. Com que tots aquests factors són massa específics i imprevisibles, la forma més directa i viable d'experimentar és utilitzar la injecció d'errors. Aquesta tècnica tot i no tenir en compte la freqüència dels errors és vàlida per determinar la robustesa davant dels mateixos.

En aquest projecte serà necessari utilitzar un llenguatge de descripció de *hardware,* els més coneguts són *Verilog* i *VHDL,* un llenguatge de programació com ara *C* i eines de simulació per tal de poder veure els resultats de les execucions, per tenir una referencia algunes plataformes de simulació conegudes són *Modelsim, i-verilog* o *Logicworks*.

3. Actors implicats

En el transcurs del projecte hi haurà un conjunt de persones implicades ja sigui directa o indirectament. Seguidament una breu explicació dels actors implicats.

3.1. Desenvolupador

És i serà el principal implicat, bàsicament s'encarregarà de desenvolupar la totalitat del projecte, tant la part tècnica, com la part de gestió del projecte i la redacció de la memòria i la documentació necessària per tal de deixar el projecte en bon estat. Treballarà sota la supervisió i la guia del director de projecte. Serà el principal responsable del progrés del projecte i el seu avanç al llarg de les diferents etapes de desenvolupament.

3.2. Director

Com que aquest projecte és purament acadèmic, és necessària la participació d'un director de projecte per tal d'orientar, guiar i aconsellar al desenvolupador que no deixa de ser l'alumne, també en cas d'haver-hi algun contratemps ajudar en la seva solució per tal de reprendre la normalitat el més aviat possible. En el cas d'aquest projecte el director és el Ramon Canal Corretger.

3.3. Beneficiaris

Els beneficiaris d'aquest projecte ho són de forma indirecta ja que aquest projecte no està orientat a usuaris finals específics tot i que en poden arribar a veure algun resultat.

El projecte està orientat a la investigació, així que com a beneficiari indirecte es pot considerar qualsevol desenvolupador o fabricant que en vulgui aplicar les conclusions. Com que es pretén determinar la implementació més fiable donada la tecnologia actual, el desenvolupador pot augmentar la fiabilitat dels seus circuits i l'usuari final pot arribar a veure una millora en el seu producte en forma de menys fallades.
A més a més, l'eina desenvolupada pot servir com a plataforma que un fabricant d'eines de suport al disseny de circuits (EDA) pugui incloure en el seu conjunt d'eines.

4.Estat de l'art

Seguidament es parlarà de les eines i el mètodes actuals implicats en aquest projecte.

4.1. Injecció d'errors

La injecció d'errors és un mètode utilitzat per tal de posar a prova els components *Hardware* de qualsevol aparell electrònic. Té com a objectiu determinar si un component o circuit és tolerant a errors de tot tipus, com ara els causats per l'impacte de partícules.

En aquest àmbit la majoria d'estudis es centren en la injecció en memòries, hi ha un bon nombre de treballs que han explotat la recerca en aquest sector, en canvi aquest estudi aplicat a circuits com ara unitats funcionals d'un processador no és tan comú.

La injecció en l'àmbit dels circuits consisteix a forçar el valor contrari al que hauria de prendre un cable, un registre o un port d'entrada/sortida d'un mòdul. Per a poder aconseguir això hi ha diverses eines experimentals i mètodes desenvolupats.

Tot i les eines disponibles, no hi ha un injector genèric que serveixi per a qualsevol circuit. La varietat de possibilitats en la implementació d'un circuit a nivell de portes lògiques, fa que sigui necessari dissenyar un injector específic per a cada circuit. Per tant, les eines disponibles intenten facilitar la implementació d'aquests injectors.

La principal divisió la marca el llenguatge de descripció de *hardware* (*HDL*) escollit per a dissenyar el circuit, principalment *Verilog* i *VHDL*, ja que l'eina que interactuarà amb el disseny ha d'estar orientada a l'utilitzat durant el disseny.

4.1.1. Injecció orientada a la simulació

Les eines d'aquest tipus estan basades en *PLI* (*Procedural Language Interface*) i proporcionen llibreries de llenguatge *C,* que permeten accedir a les dades en temps real del circuit sense necessitat de modificar ni manipular el disseny en *HDL* [3]. Això les fa molt bones per a la injecció d'errors sobre circuits ja compilats.

Com ja s'ha dit abans les eines han d'estar orientades al *HDL* utilitzat, per això dins de les eines *PLI* podem destacar, *VPI*(*verilog procedural interface)* [4], *VHPI* (*VHDL procedural interface*) [5].

En aquest tipus d'eines les més esteses i utilitzades són les orientades a *verilog*, això es deu al fet que aquest llenguatge té més similituds i és més fàcil d'integrar amb el llenguatge de programació *C*.

Dins aquest àmbit sorgeixen plataformes que integren un entorn de simulació juntament amb un conjunt de comandes per a compilació i utilització de les eines descrites, una plataforma a destacar seria *icarus-Verilog.*
Aquesta plataforma proporciona un entorn de treball còmode per tal d'utilitzar l'eina d'injecció *VPI* en la creació dels programes en *C,* compilar i enllaçar els models descrits en *verilog* i els programes d'injecció i per últim simular el comportament dels circuits.

4.1.2. Injecció *Hardware*

Aquest tipus d'injecció d'errors consisteix a augmentar el disseny del circuit amb més *hardware* específic per tal d'introduir els errors directament al sistema. A diferència de l'explicat anteriorment en aquest cas, no se simula el comportament del circuit sinó que es treballa sobre circuits físics ja fabricats, això fa que en temps sigui més ràpid veure el comportament davant la injecció d'errors. Per altra banda s'ha de tenir en compte que el *hardware* afegit pot afectar les característiques del circuit original.

4.1.3. Injecció orientada a l'emulació

A diferència de la injecció *hardware* no fa falta modificar el circuit original, ja que es fa ús de *FPGAs* (*Field Programmable Gate Array*) [6], això permet emular el comportament del circuit descrivint el hardware necessari per a la injecció. Això permet aconseguir una velocitat a l'hora d'observar el comportament similar a l'obtinguda en la injecció hardware. Per contra amb alta probabilitat s'haurà de generar múltiples descripcions per tal de poder adaptar-se a les característiques de la *FPGA* escollida per emular.

4.2. Projectes similars

Tot i existir estudis similars [7] al proposat en aquest projecte, està clar que per la incapacitat que hi ha d'arribar a obtenir un injector genèric, que tingui per objectiu estudiar el comportament de qualsevol circuit davant la injecció d'erros, podem dir que es podrà aprofitar molt poc material d'altres projectes.
Com que l'estudi que es proposa és molt concret i orientat només als sumadors d'una *CPU* s'haurà de proposar i desenvolupar una solució nova tot i poder tenir com a referència l'ús que es fa de les eines en altres projectes

4.3. Observacions respecte el projecte

En aquest projecte d'entre els tipus d'injecció explicats anteriorment s'ha escollit la injecció orientada a la simulació ja que és la més versàtil, la injecció *hardware* queda descartada automàticament ja que no es disposa de material per a fabricar circuits propis i personalitzats i la orientada a l'emulació també tot i que podria ser una extensió d'aquest projecte.

El llenguatge de descripció escollit ha sigut *Verilog,* és el llenguatge amb el qual s'han dissenyat els sumadors. Una altre eina que s'ha utilitzat ha sigut el llenguatge de programació *C* per tal d'implementar els programes d'injecció d'errors juntament amb les llibreries *VPI* (*verilog procedural interface*).

Per últim com a plataforma de simulació s'ha decidit utilitzar *i-verilog.* Com s'ha explicat prèviament la plataforma *Icarus verilog* integra l'ús de *VPI* amb simulació per tal de tenir un conjunt complet per experimentar. Amb això es pretén aconseguir implementar un injector específic per a cada tipus d'implementació de sumadors.

5. Abast del projecte

En aquest apartat es parlarà tant de l'abast d'aquest projecte com dels possibles obstacles que poden sorgir al llarg del mateix.

5.1. Abast

Per a poder arribar a obtenir bons resultats s'han de seguit diversos passos.

Primer s'ha mirat si era viable implementar utilitzant *Verilog* els diferents sumadors per a poder simular i injectar en *FPGA*. S'ha vist que això comportava circuits molt més complexos que no pas implementar amb l'objectiu de simular en ordinador. Per això l'emulació en placa s'ha descartat tot i poder ser una extensió del projecte.

Un cop implementat, s'ha integrat l'eina de simulació i la d'injecció d'errors per tal de poder obtenir els resultats. L'eina i la plataforma triats són les llibreries *VPI* juntament amb la plataforma *i-verilog*.

Per últim s'han fet les simulacions necessàries per a poder obtenir resultats significatius mitjançant estadística. Les simulacions permeten comparar el resultat obtingut a l'execució un cop injectat l'error amb el resultat correcte sense errors. Un cop obtinguts s'han analitzat per tal d'esbrinar quins tipus d'implementacions són més robustes davant l'impacte de partícules.

5.2. Possibles obstacles

- Integració entre l'eina de simulació i l'eina d'injecció:
  No hi ha un software específic per a poder injectar errors i simular de forma conjunta, per tant s'haurà d'utilitzar eines com ara *Verilog Procedural Interface* (*VPI*). Que consisteix a crear un programa en *C*, que s'utilitzarà per relacionar les etiquetes dels diferents cables del circuit descrit amb variables, i així es podrà donar un valor directament i permetre la injecció de l'error. L'ús de *VPI* permetrà ja sigui posar un valor al circuit o recuperar-ne un per tal de comprovar el resultat i veure com ha afectat l'error al conjunt. Aquest procediment pot portar problemes a l'hora d'ajuntar els dos llenguatges i fer que la injecció funcioni correctament tant pel que fa a la compilació com a l'execució.

- Temps:
  Ja que el temps per a fer aquest projecte és limitat s'haurà de veure fins a quin punt pot arribar a ser més extens o si es pot arribar a estudiar tots els tipus d'implementacions tant a escala de portes com de blocs. Es pot donar el cas que per falta de temps els plans sobre les implementacions es vegin afectats.

- Errors durant les simulacions:
  Bona part de l'estudi consistirà en llençar execucions amb un seguit d'errors definits per a les diverses implementacions, per a poder obtenir resultats significatius s'haurà de fer un gran nombre d'execucions, errors en aquestes poden comportar retards en els resultats a obtenir. Errors durant aquesta etapa poden reforçar l'obstacle del temps.

5.3. Obstacles trobats

Al llarg d'aquest projecte s'han trobat diverses dificultats i obstacles, els més importants i que han tingut conseqüències significatives en el desenvolupament ja s'havien previst en l'apartat anterior.

Tot i preveure que es podien donar, el fet d'haver d'afrontar aquests obstacles i solucionar els problemes, ha comportat canvis tant pel que fa a la planificació com a la gestió econòmica.

La integració entre l'eina de simulació i la d'injecció, va ser el primer obstacle a superar, com ja es preveia la familiarització amb l'ús de les llibreries, les crides a funcions *VPI* per tal de controlar circuits, i com s'utilitzava el pas de paràmetres entre el nivell software i el hardware descrit. Va ser una tasca més complicada el que inicialment s'esperava.

Els problemes més importants s'han donat a l'etapa de simulació, donat que les primeres sèries de simulacions han sigut de l'ordre de mil milions, el temps aproximat per a completar una d'aquestes sèries i veure resultats ha sigut d'aproximadament cinc o sis dies, això en més d'una ocasió ha retardat l'abans del projecte, ja que a partir dels resultats s'han deduït errors en la implementació que un cop corregits han comportat repetir la sèrie de simulacions.

Un altre obstacle no previst que s'ha donat durant l'etapa de simulació ha sigut el fet que l'ordinador en el qual es treballava i s'implementava tot el projecte no era capaç d'executar les sèries més grans, La solució ha sigut canviar d'entorn i demanar accés als servidors del departament d'Arquitectura de Computadors. Això ha comportat un temps extra tant per gestionar l'accés com per aprendre com funciona el sistema de clústers amb què treballa el servidor.

6. Metodologia i rigor

A causa del temps limitat per a fer aquest projecte, la metodologia de treball ha sigut de tipus incremental, és a dir, s'ha escollit començar per a fer l'estudi entre un seguit de possibles implementacions de sumadors tant a nivell de bloc com a nivell de portes. A mesura que s'avançava en la feina s'han incorporat nous dissenys de sumadors a l'anàlisi.
Seguidament es presenten els diferents tipus d'implementacions de sumadors amb les quals s'ha treballat.

6.1. Eines de treball

Les principals eines utilitzades en aquest projecte són el llenguatge de descripció, les llibreries i l'entorn de simulació.
Totes elles han fet possible la implementació i simulació al llarg del projecte.

6.1.1. *Verilog*

Una eina a destacar és el llenguatge amb el qual s'han implementat tots els sumadors i els seus mòduls, aquest és *Verilog*.
Verilog és un llenguatge de descripció de *hardware* que permet dissenyar, simular i descriure circuits digitals. Després de buscar documentació i feina en l'àmbit de la simulació en la injecció d'erros s'ha escollit per sobre d'altres llenguatges de descripció coneguts com ara *VHDL* o *ABEL*, ja que el primer està més orientat a la descripció de circuits que posteriorment s'han de portar a una placa, i el segon és un llenguatge força antic que no té la mateixa presencia que el *Verilog* en treballs relacionats en la simulació d'injecció d'errors.

6.1.2. *VPI*

L'eina que permet el control i l'acció mateixa d'injectar errors en circuits és *VPI*(*verilog procedural interface*), aquesta eina consta d'un conjunt de llibreries que fan possible mòduls descrits en llenguatge *Verilog* invocar rutines escrites en llenguatge de programació *C,* i el més important, permeten a rutines escrites en *C* interactuar amb els circuits descrits a nivell de cables i registres en temps d'execució i simulació.

6.1.3. *Icarus verilog*

Icarus verilog és una plataforma que proporciona un entorn on es pot compilar, executar i simular circuits.
La plataforma integra les llibreries *VPI* i per tant permet compilar de forma conjunta fitxers escrits en *Verilog* i en *C*.
Un cop generats els fitxers necessaris actua com a eina de simulació i permet simular els circuits i executar el codi conjuntament per tal de veure els resultats de la interacció dels dos mètodes.

S'ha escollit aquesta plataforma a causa de les seves característiques que la fan una molt bona eina per treballar en la simulació d'injecció d'errors en circuits.

6.2. Entorn de treball

El projecte s'ha desenvolupat en una distribució *Linux Mint*, on s'hi va instal·lar la plataforma *i-verilog*, en aquesta distribució s'hi han implementat tots els circuits i els programes de injecció, també s'han fet algunes simulacions de prova però per a la càrrega que suposava fer nombres molt elevats de simulacions s'ha optat per treballar de forma remota amb els servidors del departament d'arquitectura de computadors de la facultat. Així doncs les principals simulacions destinades a obtenir resultats per tal d'analitzar la fiabilitat dels sumadors s'han llençat al sistema de cues amb el qual funciona la xarxa de servidors del departament.

6.3. Implementació

Per tal de desenvolupar el projecte s'han implementat diversos sumadors i alguns d'ells de diferents formes amb l'objectiu de tenir una bona gamma d'unitats funcionals a comparar i sobre les quals mesurar la seva tolerància a errors.

6.3.1. *Ripple carry*

És la implementació més bàsica de sumador, consisteix en la concatenació dels 32 *full adders*, on a cada un es calcula la suma del bit corresponent i es passa com a entrada el *carry* de sortida al següent *full adder*.

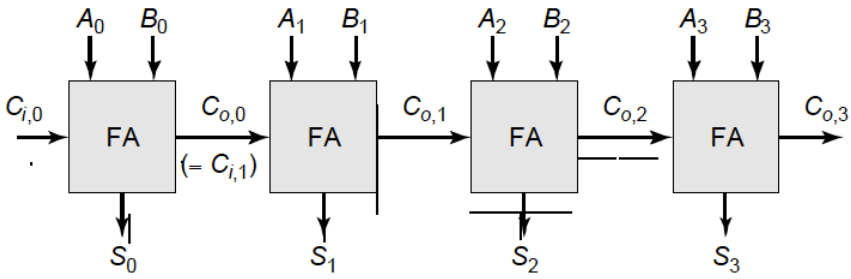

Figura 1: Sumador *ripple carry*

6.3.2. *Carry select*

Aquest sumador s'ha escollit per la similitud amb el *ripple carry*, però afegint redundància per agilitzar la propagació del *carry*. S'ha implementat agrupant els *full adders*, en grups de quatre. El grup inicial té el *carry* d'entrada original i per tant no té redundància, la resta de grups de quatre passen a ser dos grups per a calcular els mateixos bits de sortida on el primer grup té com a *carry* d'entrada 0 i l'altre 1. Un cop s'ha calculat el *carry* de sortida dels dos grups de quatre anteriors i s'ha determinat quin és el vàlid, se selecciona mitjançant un multiplexor quin dels dos paquets de sortides és el correcte.

Es preveu que aquesta redundància permeti veure certs canvis en la tolerància als errors.

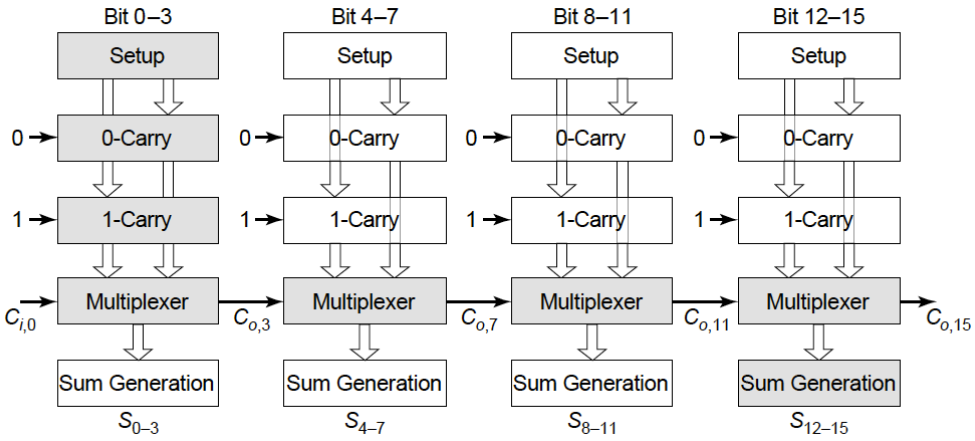

Figura 2: Sumador *carry select*

6.3.3. *Carry skip*

Per a poder provar un sumador del mateix estil que els anteriors però sense haver de dependre de *full adders*, s'ha implementat aquest sumador que consisteix en tres blocs per al càlcul de cada un dels bits de sortida. Tenim el bloc del càlcul dels bits *generate* i *propagate*, el bloc de propagació del *carry* i el bloc del càlcul de la suma.

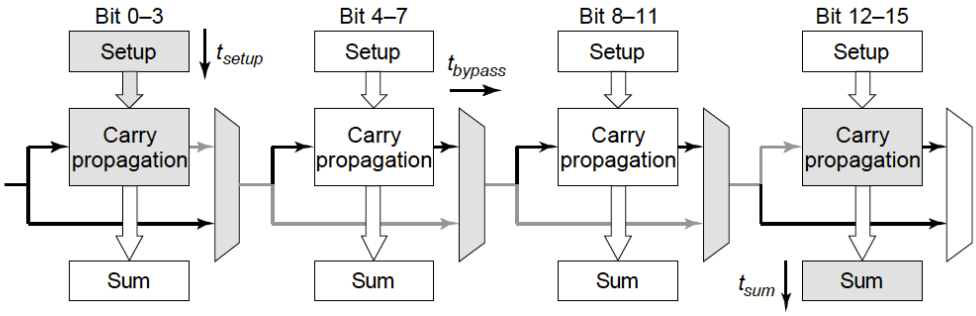

Figura 3: Sumador *carry skip*

El bloc *setup* consta de les operacions de càlcul dels bits *generate* i *propagate*.

$G_i = A_i \cdot B_i$
$P_i = A_i \oplus B_i$

En el bloc de *carry propagation* es calcula el carry de sortida i es passen les opcions cap al multiplexor.

$C_{i+1} = P_i \oplus G_i$

I per últim en el bloc *sum* es calcula la suma final dels bits d'entrada de A i B.

$S_i = G_i + C_i \cdot P_i$

6.3.4. Kogge-Stone

L'última implementació estudiada ha sigut la implementació en arbre *Kogge-Stone*. Aquesta implementació, igual que la *skip* no utilitza *full adders*, es basa també en el càlcul dels bits de *generate* i *propagate* i després propaga el *carry* en forma d'arbre fins a obtenir els resultats.
Per tal de provar aquest sumador sense haver d'utilitzar la lògica corresponent a l'arbre de 32 bits, complexa i llarga de descriure. S'ha implementat utilitzant vuit sumadors de 4 bits.

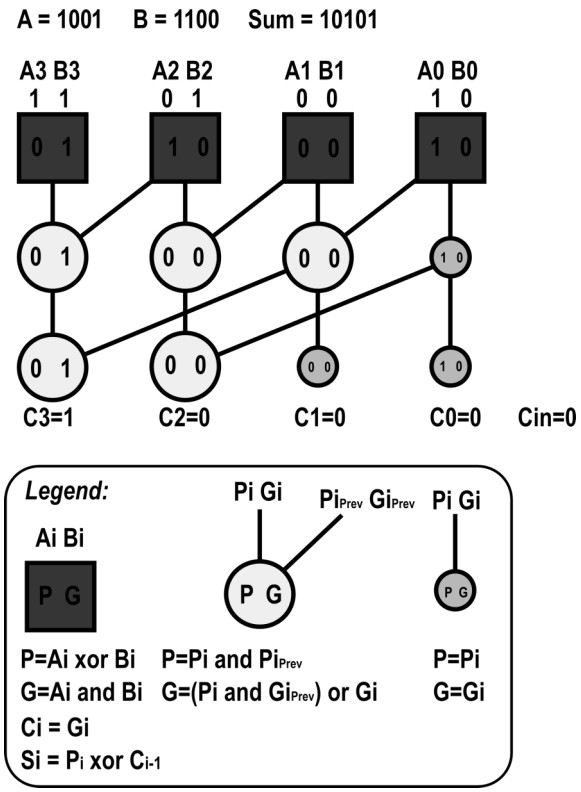

Figura 4: Funcionament d'un sumador Kogge-Stone de 4 bits

En la figura 4 podem veure un exemple de com funciona el sumador en arbre, els nodes quadrats corresponen al càlcul dels bits de *generate* i *progagate* (mòdul *setup* en el sumador *skip*), els mòduls rodons formen la lògica encarregada de propagar els *carrys*, en els més grans s'ha d'aplicar les fórmules que es mostren per tal de calcular el nou *generate* i *propagate* del node, en canvi en els petits simplement es manté els valors que arriben del node de sobre.
Tal com es veu representat, a mesura que es va calculant la propagació del *carry,* es fan càlculs a cada nivell que serveixen per alimentar nodes posteriors més propers a la sortida.

Per últim, falta calcular les sumes, els mòduls encarregats d'aquests càlculs no surten representats, però si la fórmula que han d'aplicar per tal d'obtenir cada bit de la suma final.
$S_i = P_i \oplus C_{i-1}$

6.3.5. Implementació segons portes

A més de les implementacions explicades, per els sumadors *ripple carry* i *carry select* s'han implementat els *full adders* utilitzats de diverses formes per tal d'estudiar no només l'impacte que té la implementació en la tolerància a fallades no només a nivell de bloc sinó també a nivell de portes.

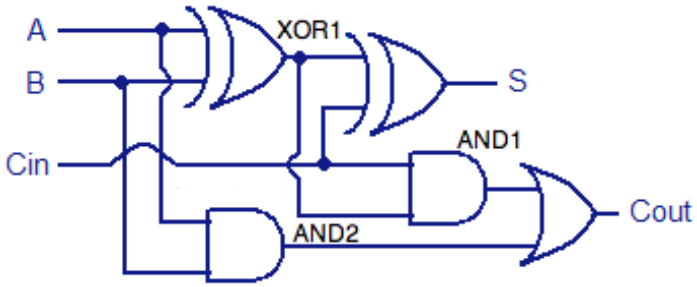

Figura 5: *Full adder* implementat amb dos portes *XOR*, dos portes *AND* i una porta *OR*

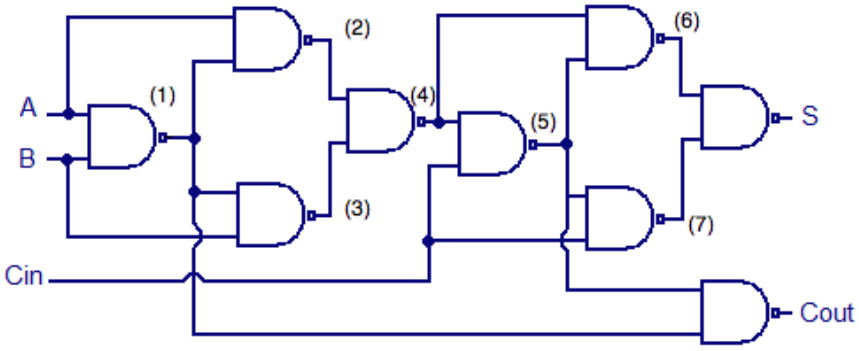

Figura 6: *Full adder* implementat amb portes *NAND*

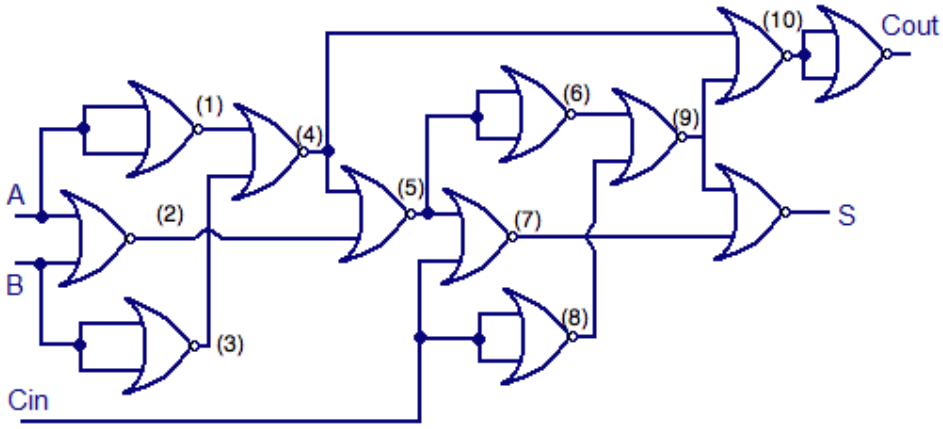

Figura 7: *Full adder* implementat amb portes *NOR*

6.3.6. Circuits destinats a la injecció

A part dels sumadors en si, s'han afegit tots els circuits necessaris per a poder simular la injecció dels errors, s'ha fet utilitzant portes *XOR* per controlar en quin punt del circuit s'injecta un error. Aquestes portes *XOR* s'han col·locat a continuació de cada una de les portes que formen el sumador.

El control funciona de la següent forma, com es pot veure a la Figura 7 si prenem la *y* com a bit de control si volem que es mantingui el mateix valor que ja teníem, només hem de fer que valgui 0 i per contra si volem que s'injecti error al cable *x* i per tant canviï de valor, fem que *y* prengui el valor 1.

Els valors que ha de prendre *y* per tal que el comportament descrit funcioni es controlen des dels fitxers escrits en llenguatge *C* que posen y = 1 com a operant a la porta *XOR* posterior a la porta del sumador on es determina que hi haurà l'error.

x	*y*	*x* XOR *y*
0	0	0
0	1	1
1	0	1
1	1	0

Figura 7: Taula de veritat de la porta XOR

Per últim el fitxer principal escrit en *Verilog* on hi ha les instancies dels sumadors i els seus mòduls també és l'encarregat de cridar el fitxer escrit en llenguatge *C* per tal d'invocar les rutines d'injecció.

Aquest fitxer és mixt i tant forma part de la descripció dels sumadors com s'encarrega de comparar els resultats obtinguts dels dos sumadors diferents instanciats. Donades les mateixes entrades, un d'ells amb injecció d'error i l'altre sense. La comparació determina si el sumador ha pogut emmascarar l'error i guarda el resultat en un dels dos registres que comptabilitzen resultats correctes i resultats incorrectes per tal de poder-ho consultar posteriorment.

Aquest fitxer utilitza el rellotge que s'hi ha descrit no només per a controlar el correcte funcionament dels sumadors sinó que també l'aprofita per a controlar les múltiples crides a les rutines d'injecció. Això permet que marcant un temps límit per cada flanc de rellotge es faci una invocació i per tant compti com una simulació, així doncs, és des d'aquest fitxer on es marca la quantitat de simulacions que es faran quan s'executi el conjunt de codi compilat.

6.3.7. Programes injectors

Un cop implementats cada un dels circuits utilitzant separació per mòduls en llenguatge *Verilog*. S'ha procedit a la implementació dels fitxers d'injecció per a cada un d'ells.

Els fitxers d'injecció consisteixen en codi escrit en llenguatge *C*, a on en primer lloc s'inicialitzen tot un seguit de punters de les llibreries *VPI* que permeten controlar els cables dels circuits, un cop indexats tots els cables necessaris, es determina aleatòriament en quin punt hi haurà un error, s'injecten les entrades generades també aleatòriament i s'injecta l'error en el punt determinat.

Com ja s'ha comentat en l'apartat anterior aquests fitxers són cridats pel fitxer principal dels circuits i per cada crida es generen noves entrades pels sumadors i un nou punt d'injecció, tot de forma aleatòria.

Per intentar mantenir el rigor, per cada un dels sumadors i les seves possibles implementacions, un cop acabada cada una d'elles s'han fet proves per comprovar el correcte funcionament. Després de comprovar que el comportament dels circuits era l'esperat s'han fet proves per veure que els errors s'injectaven en els punts que s'establien i que l'impacte que tenien era el desitjat. Un cop comprovat tot s'ha procedit a simular per tal d'obtenir resultats.

6.3.8 Càlcul de les simulacions

Per tal de que l'estudi efectuat sigui representatiu, s'ha estimat el nombre de simulacions necessari per tal d'obtenir resultats significatius.
Per fer-ho s'ha tingut en compte el nombre d'entrades i el nombre de portes, aquests dos factors determinen el nombre de possibles combinacions sobre les quals es pot donar un error i aquest pot emmascarar-se o no.
Comptant que tenim $2^{32} * 2^{32}$ * numero de portes del sumador a estudiar possibles combinacions, estem parlant de gairebé 10^{20} possibles combinacions. Aquest número de simulacions és inabastable. Per tant, s'ha fet un estudi de diferents números de simulacions per tal de garantir que el resultat final s'estabilitza.

7. Planificació temporal

7.1. Especificació de les tasques

El temps establert per fer aquest projecte és d'aproximadament 4 mesos, compresos entre febrer i juny de 2015. Seguidament es presenta una planificació de les tasques i el contingut del projecte. S'ha de tenir en compte que és una estimació i que a mesura que es vagi treballant algunes coses segurament s'hauran de modificar.

7.1.1. Gestió del projecte

Aquesta tasca compren tota la feina de documentació que demana l'assignatura de gestió de projectes (GEP) .
La feina es divideix en 7 entregues amb temps de dedicació definits de la següent forma:

- Abast del projecte (9.25 hores).
- Planificació temporal (8.25 hores).
- Gestió econòmica i sostenibilitat (9.25 hores).
- Presentació preliminar (6.25 hores).
- Contextualització i bibliografia (15.25 hores).
- Plec de condicions (12.5 hores).
- Presentació oral i document final (18.25 hores).

Els recursos necessaris per aquesta tasca són les eines ofimàtiques habituals (editor de text, visualitzador de pdf, editor de presentacions, etc), un generador de diagrames de *Gantt* i eines de comunicació com ara el Racó de la FIB i Atenea.

7.1.2. Familiarització amb l'eina d'injecció d'errors

En aquesta tasca el que es pretén aconseguir, és aprendre a utilitzar l'eina d'injecció d'errors. Com que l'eina consisteix a crear un programa en llenguatge *C* per a cada circuit, seguit determinats patrons i utilitzant determinades funcions i estructures. És important aprendre com crear aquests programes amb certa facilitat i comoditat, ja que serà una de les feines més repetides durant la tasca d'implementació.
També serà important experimentar com es comporta l'eina de simulació un cop estigui integrat el circuit amb el programa creat per a injectar els errors.
Els recursos necessaris seran l'editor escollit per a programar i la documentació trobada en cercar sobre l'eina.

7.1.3. Avaluació del mètode d'implementació

És important abans de començar a implementar tenir calar i haver analitzat bé quin mètode es farà servir. Aquesta tasca consistirà a fer l'anàlisi. S'ha de veure si el projecte s'implementarà per a funcionar sobre una *FPGA* o si les execucions es faran sobre simulador en el nostre cas el *Modelsim*. En cada cas la implementació varia molt, si treballem sobre *FPGA* s'ha de tenir en compte que els circuits passen a ser molt més complexos en haver d'explicitar com a entrades els punts on es podran introduir els errors, cosa que t'estalvies en la implementació orientada al simulador. S'haurà de veure si és viable i no porta problemes escollir-ne una o l'altre fent les proves pertinents.
En aquest cas els recursos utilitzant seran, el mateix editor de *Verilog* del simulador *Modelsim* i els programes de prova implementats en *C* i un compilador de *C*.

7.1.4. Implementació dels circuits i dels programes d'injecció

Aquesta tasca consistirà bàsicament en la implementació general del projecte.
Es començarà amb cada un dels tipus de sumadors triats: *Ripple carry, carry select, carry skip, Look-ahead, Kogge-Stone* i per a cada un d'ells s'implementarà de tres formes diferents, utilitzant portes *ANDs* i *XORs*, només portes *NANDs* i per últim només portes *NORs*. Un cop implementats es procedirà a comprovar que funcionen correctament mitjançant el simulador i comparant les sortides amb els resultats esperats.
El següent a implementar serà per cada un dels circuits previs, un programa en *C* estructurat i utilitzant l'eina d'injecció d'errors. En aquest cas la comprovació que funciona correctament es farà mentre se simula i es seguirà l'error mitjançant etiquetes prèviament col·locades en implementar els circuits.
Els recursos necessaris un cop més seran els editors tant de *C* com de *Verilog* així com el simulador *Modelsim* o en cas de ser necessari una *FPGA*.

7.1.5. Simulacions

Aquesta serà la tasca on primer, es buscarà quin és el nombre de simulacions necessari per a poder obtenir conclusions utilitzant mitjans estadístics. Un cop determinat aquest nombre es procedirà a determinar quins seran els jocs de proves, és a dir, quins seran els errors que s'injectaran i on s'injectaran. Per últim s'executaran els jocs de proves i es simularan els resultats.
En aquesta tasca farà falta el *Modelsim* i un ordinador disponible per funcionar durant moltes hores seguides.

7.1.6. Tractament dels resultats

Aquesta tasca es podrà realitzar paral·lelament a les simulacions. A mesura que es vagin obtenint els resultats de cada una de les implementacions es pot anar consultant en taules estadístiques i fer els càlculs immediats necessaris per a anar obtenint conclusions.
Els recursos necessaris en aquesta tasca seran bàsicament taules estadístiques.

7.1.7. Finalització

Aquesta tasca consistirà a acabar el projecte, redactar la memòria definitiva i fer i assajar la defensa davant el tribunal.
Per aquesta tasca serà necessari disposar de les eines ofimàtiques habituals.

7.2. Temps dedicat a cada tasca

Tasca	Estimació del temps en hores
Gestió del projecte	79
Familiarització amb l'eina d'injecció d'errors	25
Avaluació del mètode d'implementació	80
Implementació dels circuits i els programes d'injecció	200
Simulacions	150
Tractament dels resultats	60
Finalització	60
Total	654

Taula 1: Temps per tasca

7.3. Diagrama de Gantt

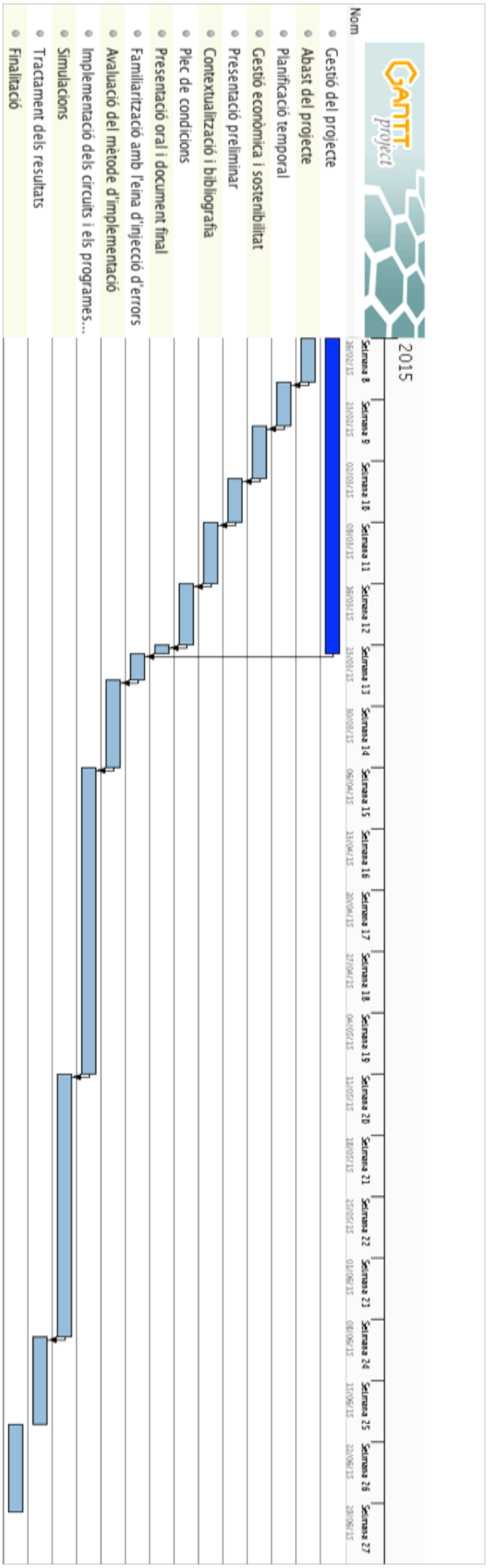

Figura 8: Diagrama de Gantt

7.4 Recursos

A continuació es descriuen els recursos *hardware*, *software* i humans necessaris pel correcte desenvolupament d'aquest projecte.

7.4.1 Recursos Hardware

- Ordinador personal: es farà servir un portàtil MacBook Pro per a la documentació, la implementació i les simulacions de prova.
  Durant les simulacions es pot donar el cas que l'ordinador proposat no sigui prou potent en aquest cas es demanaria per utilitzar els servidors del departament de computadors de la *FIB*.

- El hardware del clúster on s'ha simulat és el següent:

  - Nodes any 2014 (arvei-1401 .. arvei-1440)
    - *40 Nodes Xeon E5-2630L v2 a 2.40GHz*
    - *128 GB memòria RAM*
    - *1 disc dur d'1 TB SATA-3*
    - *4 Targetes de xarxa Intel Gigabit Ethernet*
  - Nodes any 2012 (arvei-114 .. arvei-153)
    - *40 Nodes USP Xeon E5-2630L a 2GHz*
    - *64 GB memòria RAM*
    - *1 disc dur d'1 TB SATA-3*
    - *4 Targetes de xarxa Intel Gigabit Ethernet*
  - Nodes any 2010 (arvei-74 .. arvei-113)
    - *40 Nodes USP Xeon L5630 Dual*
    - *Placa base Intel S5520URR amb 6 canals de memòria DDR3*
    - *xassís Intel SR1600*
    - *Chipset Intel 5520 amb I/O Controller Hub ICH10R*
    - *2 processadors Intel Xeon Quad-Core L5630 @ 2.13GHz*
    - *24 GB memòria RAM en 6 mòduls de 4GB*
    - *2 discs durs de 320 GB SATA-2*
    - *2 Targetes de xarxa Intel Gigabit Ethernet*
  - Nodes any 2006 (arvei-1 .. arvei-73)
    - *73 Nodes USP Xeon Dual-Core 5148*
    - *Placa base Intel S5000VCL*
    - *xassís Intel SR1530*
    - *Chipset Intel 5000V*
    - *2 Processadors Intel Xeon Dual-Core 2.333GHz, FSB 1333MHz, 4MB Cache*
    - *12 GB Memòria RAM en 6 mòduls de 2 GB*
    - *Disc dur SEAGATE Barracuda 320 GB S-ATA-2*
    - *2 Targetes xarxa Intel Pro/1000 Gigabit Ethernet*

- Càmera de vídeo utilitzada per a gravar la presentació demanada a la tasca Presentació preliminar dins la Gestió del projecte.

7.4.2 Recursos Software

Les eines *software* que s'utilitzaran al llarg d'aquest projecte són les següents:

- *i-verilog* : plataforma per compilar i simular de circuits descrits en *Verilog.*
- *Sublime Text*: editor de text orientat a la programació.
- *Microsoft Office*: per fer tota la documentació i les presentacions.
- *Verilog procedural interface* (VPI): eina per programar la injecció d'errors.
- Visualitzador de *PDF*s: per a poder visualitzar documentació.
- *Dropbox, Google Drive*: tenir còpia dels documents i els arxius del projecte i poder editar *online*.
- *Ganttproject*: eina per crear diagrames de *Gantt.*

7.4.3 Recursos Humans

Aquest projecte només el desenvoluparà l'estudiant que el cursa amb la guia del seu director, no hi intervenen tercers d'una empresa o similars.

7.5 Desviacions i pla d'actuació

Al llarg del projecte ens podem desviar respecta la planificació inicial.
Com que no és fàcil estimar el temps necessari per cada una de les tasques, i tampoc es pot preveure els inconvenients que puguin sorgir i les conseqüències sobre el temps que puguin provocar. S'ha de planificar les possibles solucions a aplicar.

El pla d'actuació seria aprofitar el mètode de treball incremental que permet aquest projecte i tenir en compte el punt crític de la tasca d'implementació:

- Si per algun motiu una tasca tarda més del planificat es continua amb ella fins a finalitzar, si es veu que el temps perdut total en sumar les tasques que sobrepassen el temps assignat és excessiu i pot perjudicar la finalització del projecte, i ens trobem a un put abans de la tasca d'implementació. La implementació per simular sobre *FPGA* es descarta.

- En cas d'haver sobrepassat l'etapa d'implementació, donada la mateixa situació crítica. Es continua amb el mateix volum de feina i s'intenta obtenir més ordinadors per reduir el temps de la tasca de simulació. En el cas que no es poguessin aconseguir, es procediria a descartar implementacions sobre les quals simular, intentant mantenir el volum de feina proper a l'original.

Aquestes desviacions i la seva correcció comporten un augment dels recursos en un dels seus casos, com s'ha descrit farien falta més ordinadors.

El pla d'actuació proposat i les reunions periòdiques amb el director del projecte, garanteixen amb certa seguretat la finalització del projecte dins els seu termini.

7.6 Modificacions en la planificació

A causa dels obstacles trobats al llarg del desenvolupament del projecte, la planificació inicial s'ha vist afectada principalment en concepte de temps i per tant en recursos humans.

La primera etapa afectada ha sigut la de familiarització amb l'eina d'injecció, s'ha invertit més hores per tal d'aprendre a utilitzar tot l'entorn de descripció, implementació i simulació.

L'augment en temps més considerable ha sigut en les etapes d'implementació i simulació. La principal causa ha sigut els problemes que comportava tenir petits errors en la implementació que es reflectien en els resultats de les simulacions, forçant no només haver de corregir l'error sinó també repetir simulacions que podien arribar a tardar una entre cinc dies i una setmana sencera.

8. Gestió econòmica

8.1. Pressupost

	Nº	**Preu unitat (€)**	**Vida útil (anys)**	**Amortització estimada (€)**	**Preu (€)**
Costos directes					
Gestió del projecte					
Ordinador(MacBook Pro)	1	1300	5	12.64	12.64
Càmera de vídeo	1	230	3	2.21	2.21
Ganttproject	1	0	-	0	0
Visualitzador PDF	1	0	-	0	0
Microsoft office	1	99	1	0.95	0
Recursos humans	79	30	-	0	2370
Familiarització amb l'eina					
Ordinador(MacBook Pro)	1	1300	5	4	4
Editor	1	0	-	0	0
Plataforma iverilog	1	0	-	0	0
Recursos humans	25	30	-	-	750
Avaluació del mètode d'implementació					
Ordinador(MacBook Pro)	1	1300	5	12.8	12.8
Editor	1	0	-	0	0
Plataforma iverilog	1	0	-	0	0
Recursos humans	80	30	-	-	2400
Implementació					
Ordinador(MacBook Pro)	1	1300	5	32	32
Editor	1	0	-	0	0

Plataforma iverilog	1	0	-	0	0
Recursos humans	200	30	-	-	6000
Imprevistos	40	35% (10.5)			420
Simulacions					
Ordinador(MacBook Pro)	1	1300	5	24	24
Servidor del departament	1	4000	3	0	0
Plataforma iverilog	1	0	-	0	0
Recursos humans	150	30	-	-	4500
Imprevistos	30	20% (6)			180
Tractament de resultats					
Ordinador(MacBook Pro)	1	1300	5	9.6	9.6
Microsoft office	1	99	1	0.72	0.72
Recursos humans	60	30	-	-	1800
Finalització					
Ordinador(MacBook Pro)	1	1300	5	9.6	9.6
Microsoft office	1	99	1	0.72	0.72
Visualitzador PDF	1	0	-	0	0
Recursos humans	60	30	-	-	1800
Costos indirectes					
Llum	4	10			40
Internet	4	11			44
Paper	1	10			10
Total acumulat					20422.29
contingències		10% del total			2042.23
Total sense IVA					22464.52
Total amb IVA(21%)					26058.84

Taula 2: Pressupost

El pressupost s'ha desglossat en costos directes per cada una de les tasques del diagrama de *Gantt* i costos indirectes.
A cada una de les tasques es contemplen els recursos *hardware*, *software* i humans.

Per cada un dels recursos tant de *hardware* com de *software* que tenen un cost, s'ha calculat l'amortització tenint en compte les hores de funcionament i la seva vida útil. Es pot veure que molt del material en *software* utilitzat és lliure, i per tant no implica cap mena de despesa pel projecte.

En l'apartat de simulacions es preveu que s'utilitzarà el servidor del departament, un cop més la seva utilització no reporta despeses pel projecte i no se'n calcula l'amortització.

Els imprevistos s'han volgut reflectir en les dues etapes més importants i per tant més propenses a tenir-ne, les d'implementació i simulació. El percentatge de risc s'estima segons la importància de la tasca i la seva dificultat, en el cas de l'etapa d'implementació un 35% de risc respecte del preu d'una hora de feina i en l'etapa

de simulació un 20%. Les hores comptabilitzades per imprevistos venen en funció de la mateixa tasca deixant un 20% de les hores inicials.

Els únics costos indirectes que són rellevants pel projecte són els de la quota mensual de llum i de connexió a internet així com la despesa en paper a l'hora d'imprimir la memòria del projecte.

El percentatge de contingència per aquest projecte s'ha estimat d'un 10% per tal de tenir un marge prudent al final de projecte.

8.2. Pla de control

Per tal de tenir controlades les possibles desviacions al llarg del projecte, el millor que es pot adoptar com a pla de control és revisar les despeses reals al final de les etapes més importants. Així doncs, es procedirà a fer la comprovació al final de les etapes d'implementació i simulació. Si hi ha hagut desviacions respecte de la planificació es mirarà si queden cobertes amb el marge donat pels imprevistos. En cas contrari s'hauria d'estudiar per quins motius la desviació ha estat tan gran per tal de no tornar a cometre el mateix error en una altra etapa o un futur projecte.

Per últim al final del projecte s'ha de fer un estudi per mirar si les despeses finals cauen dins els marges previstos i si el marge de contingències és suficient.

Amb aquest mètode i tenint en compte que es estimacions han estat força acurades i reals, també que molts dels recursos a utilitzar són gratuïts i que s'ha deixat bons marges de seguretat, amb molta probabilitat els costos reals no s'allunyaran dels del pressupost.

8.3. Desviacions del pressupost

Un cop acabat el projecte i veien els obstacles que han sorgit i les desviacions en la planificació. Si tenim en compte que l'augment d'hores pel que fa a recursos humans veiem que hi poden haver modificacions en el pressupost.
En el pressupost estan comptabilitzades entre imprevistos i contingències 84 hores més de recursos humans així que l'augment dels costos seria de comptabilitzar unes 70 hores més de treball.

9. Sostenibilitat del projecte

En el següent estudi es miraran els aspectes social, ambiental i econòmic. Per a comprovar la sostenibilitat en cada un dels punts es procedirà a respondre un seguit de preguntes de les quals se'n deduirà una certa puntuació.

9.1. Dimensió econòmica

Els punts tinguts en compte per tal de justificar la sostenibilitat econòmica són els següents:

- Els recursos tant materials com humans i els seus costos s'han tingut en compte i s'han avaluat en l'apartat de Gestió econòmica mitjançant un pressupost.
- La base del projecte és fonamentalment investigació, per tat, no es treballa sobre un producte o servei que necessiti manteniment i per tant no suposa cap despesa.
- El pressupost del projecte és ajustat i això pot fer que sigui competitiu en cas d'estar orientat en aquest sentit.
- Es podria fer el mateix projecte amb els mateixos recursos materials però amb menys temps i per tant menys recursos humans només en el cas que es realitzés per un enginyer experimentat.
- El temps a cada una de les tasques és proporcional a la seva complexitat i sobretot als possibles contratemps que poden sorgir, això el fa sòlid, ja que hi ha marge de maniobra suficient.

9.2. Dimensió social

Els punts tinguts en compte per tal de justificar la sostenibilitat social són els següents:

- La situació social és de crisi econòmica estesa amb possibilitat de millora a relatiu curt termini, la situació política és estable. El sector al qual pertany el projecte és un sector a l'alça amb moltes oportunitats.
- L'envergadura d'aquest projecte no arriba ni molt menys a poder influir a una escala tan gran com ara la situació del país
- La necessitat d'aquest projecte és tan de l'àmbit de la investigació com de l'empresari que vol augmentar la fiabilitat dels seus productes, el projecte pretén investigar sobre una temàtica força important dins l'àmbit informàtic, però no serà l'únic material existent i per tant la seva necessitat és relativa.
- Satisfer aquesta necessitat pot ajudar a determinar com fer més fiables alguns productes de cara a consumidors finals.
- El consumidor que es pot veure afectat pels resultats d'aquest projecte en veuria els efectes mitjançant fiabilitat dels productes.
- El projecte no afecta negativament a cap col·lectiu.

9.3. Dimensió ambiental

Els punts tinguts en compte per tal de justificar la sostenibilitat ambiental són els següents:

- El principal recurs a utilitzar que pot tenir un impacta en el mediambiental és la llum, i aquest impacte és totalment indirecte. Un altre recurs seria el paper necessari per a la impressió de la memòria, un cop més una despesa poc important.
- El projecte en si no afecta molt mediambientalment així que el fet que no es fes no seria rellevant ni significaria cap millora.
- En aquest projecte no es treballa amb matèria primera així que, ni es pot reciclar material físic d'altre projectes, ni possiblement se'n pugui aprofitar res d'aquest.
- La contaminació que pot arribar a produir serà de forma indirecta deguda a la producció de la llum necessària per al funcionament dels ordinadors necessaris.
- No influirà ni en més ni en menys en l'empremta ecològica.

9.4. Matriu de puntuació

Després de l'estudi fet en els apartats anteriors s'estima la següent puntuació:

Sostenible?	Econòmica	Social	Ambiental
Valoració	8	6	7

Taula 3: Puntuació sostenibilitat

10. Resultats

En aquest apartat s'analitzaran i es comentaran els resultats obtinguts de les simulacions realitzades.

10.1. Estudi inicial

Inicialment, abans de començar a simular, s'ha analitzat la lògica de cada una de les implementacions per portes amb les quals s'ha treballat, l'objectiu d'aquest anàlisi és tenir una estimació prèvia del que es podria veure en els resultats finals. També ha ajudat a determinar si els resultats que s'anaven obtenint estaven dins d'uns marges raonables i es podien considerar representatius i vàlids.

A	B	c_in	xor1	and1	and2	s	cout
0	0	0	0	0	0	0	0
0	0	1	0	0	0	1	0
0	1	0	1	0	0	1	0
0	1	1	1	1	0	0	1
1	0	0	1	0	0	1	0
1	0	1	1	1	0	0	1
1	1	0	0	0	1	0	1
1	1	1	0	0	1	1	1

Taula 4: Comportament lògic del primer disseny de *full adder* (disseny convencional)

A	B	c_in	nand1	nand2	nand3	nand4	nand5	nand6	nand7	s	cout
0	0	0	1	1	1	0	1	1	1	0	0
0	0	1	1	1	1	0	1	1	0	1	0
0	1	0	1	1	0	1	1	0	1	1	0
0	1	1	1	1	0	1	0	1	1	0	1
1	0	0	1	0	1	1	1	0	1	1	0
1	0	1	1	0	1	1	0	1	1	0	1
1	1	0	0	1	1	0	1	1	1	0	1
1	1	1	0	1	1	0	1	1	0	1	1

Taula 5: Comportament lògic de disseny amb portes *NAND*

A	B	c_in	nor1	nor2	nor3	nor4	nor5	nor6	nor7	nor8	nor9	nor10	s	cout
0	0	0	1	1	1	0	0	1	1	1	0	1	0	0
0	0	1	1	1	1	0	0	1	0	0	0	1	1	0
0	1	0	1	0	0	0	1	0	0	1	0	1	1	0
0	1	1	1	0	0	0	1	0	0	0	1	0	0	1
1	0	0	0	0	1	0	1	0	0	1	0	1	1	0
1	0	1	0	0	1	0	1	0	0	0	1	0	0	1
1	1	0	0	0	0	1	0	1	1	1	0	0	0	1
1	1	1	0	0	0	1	0	1	0	0	0	0	1	1

Taula 6: Comportament lògic del disseny amb portes *NOR*

A les taules anteriors s'analitzen totes les possibles combinacions d'entrades i sortides per a cada una de les implementacions segons les portes utilitzades.

Les columnes intermèdies mostren els valors que prenen les sortides de cada una de les portes que conformen el sumador donades unes entrades especifiques. La numeració que s'utilitza per etiquetar les portes és la mateixa que es mostra en els esquemàtics dels dissenys mostrats en les Figures 4 5 i 6.

A cada una de les taules es mostra amb groc els cables sobre els quals corresponen els valors d'aquella columna, en taronja tenim els possibles valors que poden prendre les entrades, en vermell tots els punts del circuit on si s'hi injectés un error provocaria un mal funcionament i per últim en verd els punts del circuit on si s'injectés un error degut el comportament de les portes de cada un dels dissenys l'error quedaria emmascarat, per tant el sumador toleraria l'error i les sortides serien correctes.

	Convencional	NAND	NOR
Emmascarament(%)	10	14	22

Figura 9: Percentatge d'emmascarament teòric

Si calculem els percentatges que ens mostren les taules dividint els punts del circuit que toleren un error pel total de punts on es podria injectar obtenim els valors de la Figura 9 i veiem a priori que analíticament la fiabilitat davant errors del disseny amb portes NOR(Figura 6) és la més alta seguit del disseny amb portes *NAND*(Figura 5) i per últim el disseny convencional (Figura 4).

10.2. Comparativa entre els dissenys

Seguidament es compararan els resultats obtinguts de les simulacions i es farà un anàlisi de les diferents implementacions en diversos aspectes.

Es vol mirar l'emmascarament i per tant la tolerància a errors de cada implementació, així com l'afecta que tenen les dimensions del sumador en els errors, l'evolució dels resultats respecte el nombre de simulacions i el temps d'aquestes.

10.2.1. Simulacions i emmascarament

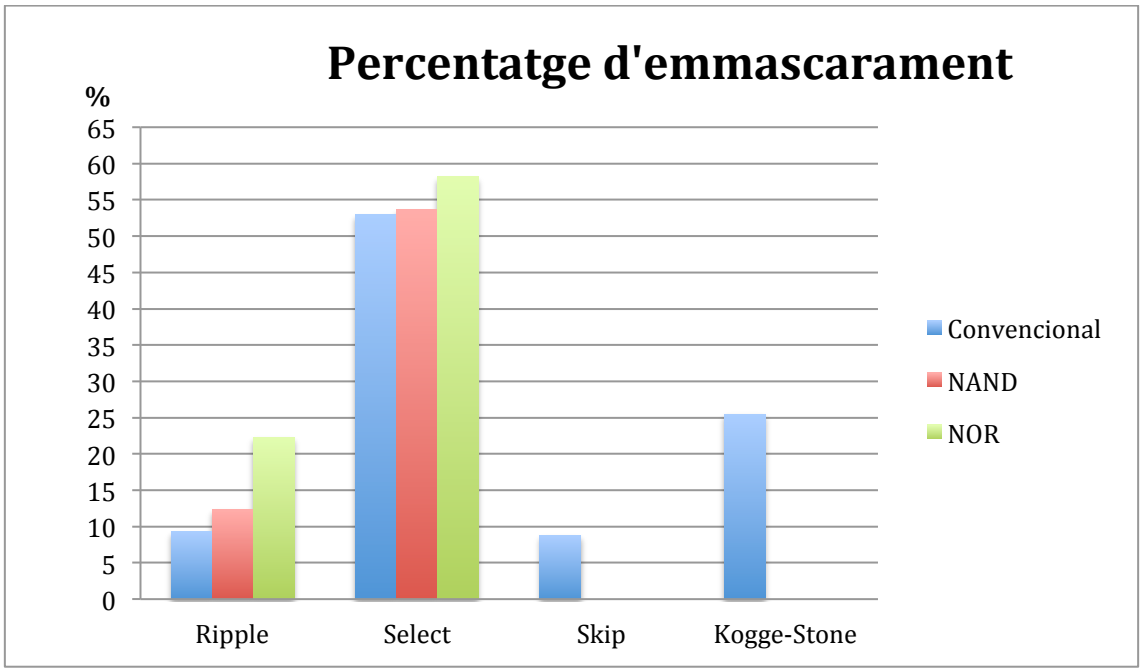

Figura 10: Percentatges d'emmascarament

En la figura anterior és mostren els resultats obtinguts en les series de simulació de mil milions per sèrie, el sumador Kogge-Stone la sèrie de cent milions. Veiem que els resultats respecte la implementació per portes són similars als vistos en l'estudi previ.

La gràfica mostra els tres grups de sumadors pel que fa a blocs, *ripple*, *select* i *skip* i pels dos primers les seves implementacions segons les portes utilitzades, tenim la implementació convencional, amb portes *NAND* i amb portes *NOR*, Figures 4, 5 i 6 respectivament.

El primer que podem veure és que en el sumador *ripple*, la implementació per portes més robusta és amb *NOR*. Això és degut al comportament de la pròpia porta ja que donat un 1 a una de les seves entrades la sortida és sempre igual sense importar l'altre entrada, això i el nombre de portes concatenades permeten un millor emmascarament. Aquest comportament ja s'havia observat a la taula 6. La implementació amb portes *NAND* és la següent més robusta, això sorprèn ja que aquesta implementació té la mateixa particularitat que l'explicada per la *NOR,* cada porta té 3 combinacions a un valor i 1 a un altre. Intuïtivament haurien de tenir un percentatge d'emmascarament similar, però al comprovar els resultats s'ha vist que no. Per últim va la convencional que implementa amb tres tipus diferents de portes que és la menys tolerant a fallades.

El mateix que hem vist amb el sumador *ripple* passa amb el sumador *select*, veiem que es manté l'ordre de tolerància entre les implementacions a nivell de porta tot i que la diferència es redueix molt, això pot ser degut a la redundància que incorpora el sumador *select* que fa que s'igualin les toleràncies.

Si mirem les tres implementacions a nivell de bloc, veiem que els percentatges d'emmascarament més alts estan en la implementació *select*, seguidament ve la implementació *ripple* i per últim l'*skip*.
Com hem explicat abans el select té redundància respecte del *ripple* i per tant només de partida té un 50% de probabilitats que l'error es doni sobre un sumador que no se n'utilitzarà el resultat i això li dóna molta robustesa.

També tenim el *carry skip*, aquest sumador en implementar-se de diferent forma i no utilitzar *full adders* és més vulnerable als erros, aquest sumador també té una porta *OR* que com hem vist en el *ripple* convencional és on s'emmascara el problema és que l'error que li arriba a la porta si ve donat per un canvi en el bit *propagate* o *generate* significa que l'error també arriba a altres portes i per tant hi ha menys probabilitats d'emmascarament.

Per últim, veiem que el sumador *Kogge-Stone* tot i que també té la necessitat de tenir els bits *propagate* i *generate* correctes en tot moment, és més tolerant a fallades no només que el sumador *skip*, sinó també que totes les variants de *ripple.*
Això pot ser degut a la propagació en arbre que té, cada nivell de l'arbre rep resultats d'un dels nivells superiors per tal de generar els bits corresponents a aquell nivell i les operacions per tal de fer aquests càlculs impliquen més portes *AND*.
Cada sumador dels vuit que formen el principal té setze portes *AND.*
Si tenim en compte que moltes operen amb el bit *generate*, i que aquest bit es calcula amb la *AND* dels dos bits d'entrada, veiem que hi ha un quart de possibilitats que alguna de les portes *AND* de l'arbre sigui capaç d'emmascarar un error fixat un zero a l'altra entrada.

10.2.2. Emmascarament per mida del sumador

S'ha de tenir el compte que la tolerància als errors que estem estudiant es veu afectada directament per la mida del sumador i per tant per el nombre de portes que el componen. Com més gran és el sumador més àrea hi ha on pot haver un impacte de partícules que causi l'error. La següent figura mostra el percentatge d'emmascarament del circuit dividit pel número de portes que l'integren.

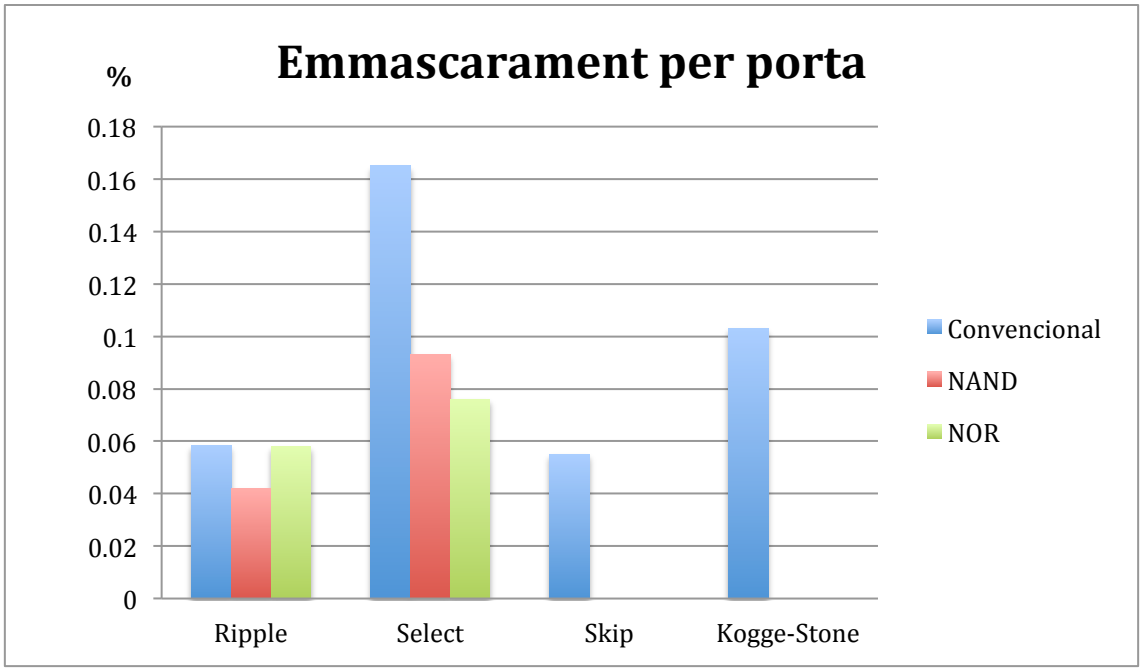

Figura 11: Emmascarament per porta

Sabem que el nombre de portes utilitzat per implementar cada sumador creix en la implementació *NAND* comparat amb la convencional i en la *NOR* respecte les altres dues.

En el gràfic anterior podem veure com a diferència d'abans en el sumador *ripple*, la implementació convencional té millor o igual percentatge d'emmascarament per cada una de les seves portes que les altres dues implementacions.
Això ens diu que el guany en emmascarament que obtenim en la implementació *NOR* és suficient per compensar l'augment de portes utilitzades, en canvi tot i l'augment d'emmascarament de la implementació *NAND* respecte la convencional, no és suficient per a compensar l'augment en àrea.

En la implementació *Select,* veiem que l'augment en portes segons la implementació afecta molt més, ja que la redundància fa que hi hagi el doble de portes que el *ripple,* tot i això veiem que l'emmascarament per porta de la implementació *select* és la més alta, concretament la convencional, que es beneficia de la redundància i evita l'augment de portes que suposen les altres dues implementacions per portes.

Si mirem la implementació skip veiem que tot i tenir el mateix nombre de portes que el *ripple* normal té un emmascarament per porta més baix, això ve del fet que aquesta implementació és la que té el percentatge d'emmascarament més baix i per tant tot i tenir menys portes la tolerància segona la mida disminueix.

Finalment veiem que la implementació *Kogge-Stone* és la segona amb millor percentatge d'emmascarament per porta, això és així, ja que és la quarta de les vuit en emmascarament però alhora és la segona que menys portes utilitza després de

la *ripple* convencional. Això mostra que tot i no ser la més tolerant de totes les implementacions, sí que per l'augment d'espai pel que fa a portes utilitzades respecte a la implementació més reduïda s'obté una molt bona millora en la tolerància a fallades del sumador.

10.2.3. Nombre de simulacions

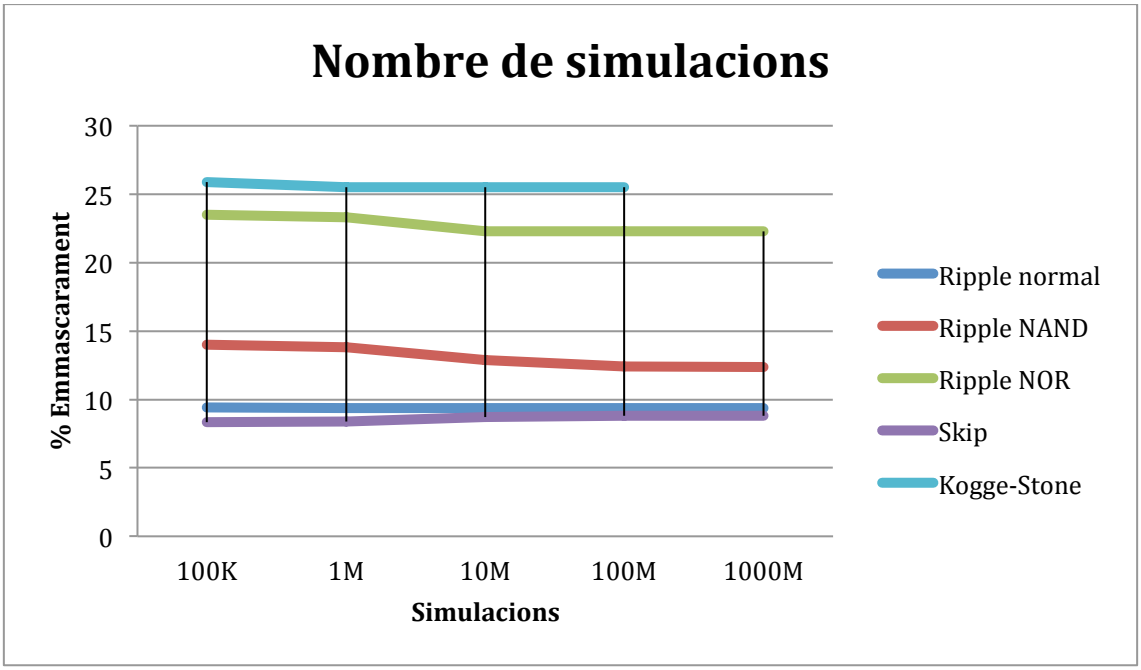

Figura 12: Nombre de simulacions disseny ripple

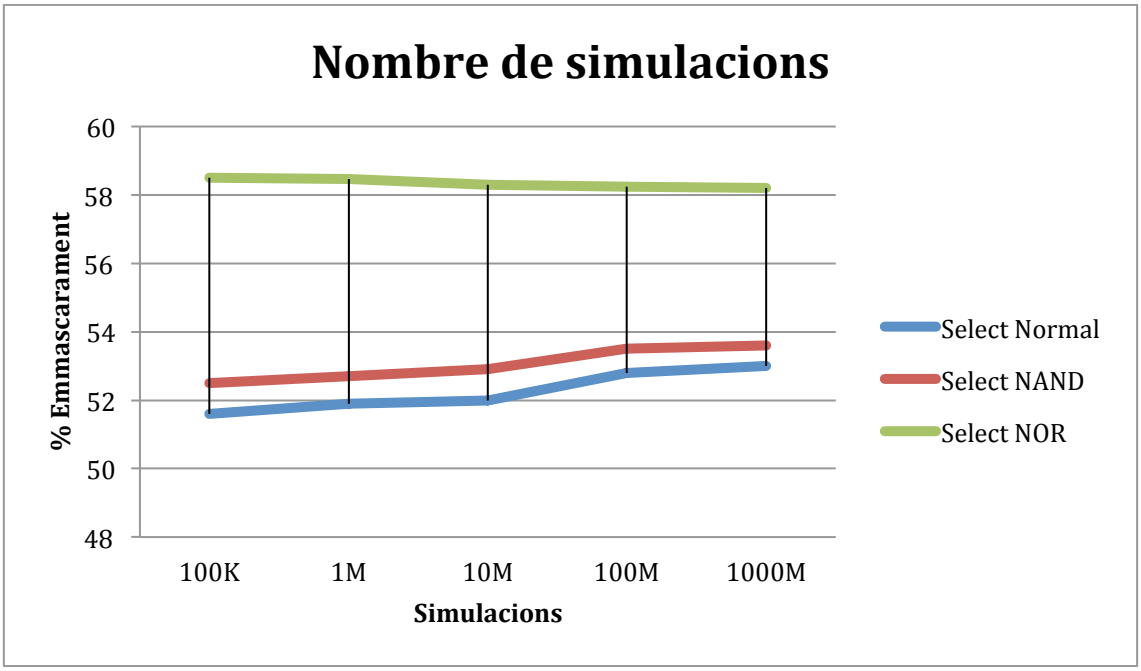

Figura 13: Nombre de simulacions disseny select

A les figures 12 i 13 es pot veure que el percentatge d'emmascarament calculat s'estabilitza a partir dels 100 milions de simulacions per tots els dissenys trobats. I per tant ha sigut el valor escollit per utilitzar-ne els resultats en la comparativa entre les implementacions. La diferència màxima entre el valor calculat en 100K simulacions i 1000M és al sumador *Select* amb implementació normal. Aquesta diferència és 1.1% en valor absolut.

10.2.4. Temps de simulació

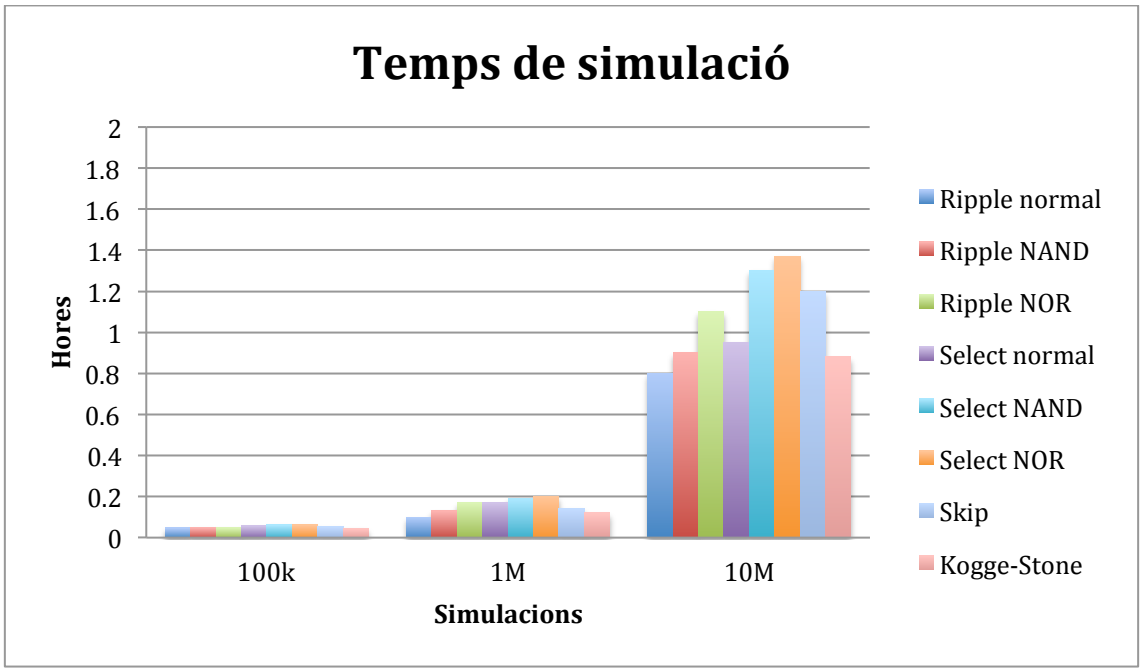

Figura 14: Augment del temps de 100 mil a 10 milions de simulacions

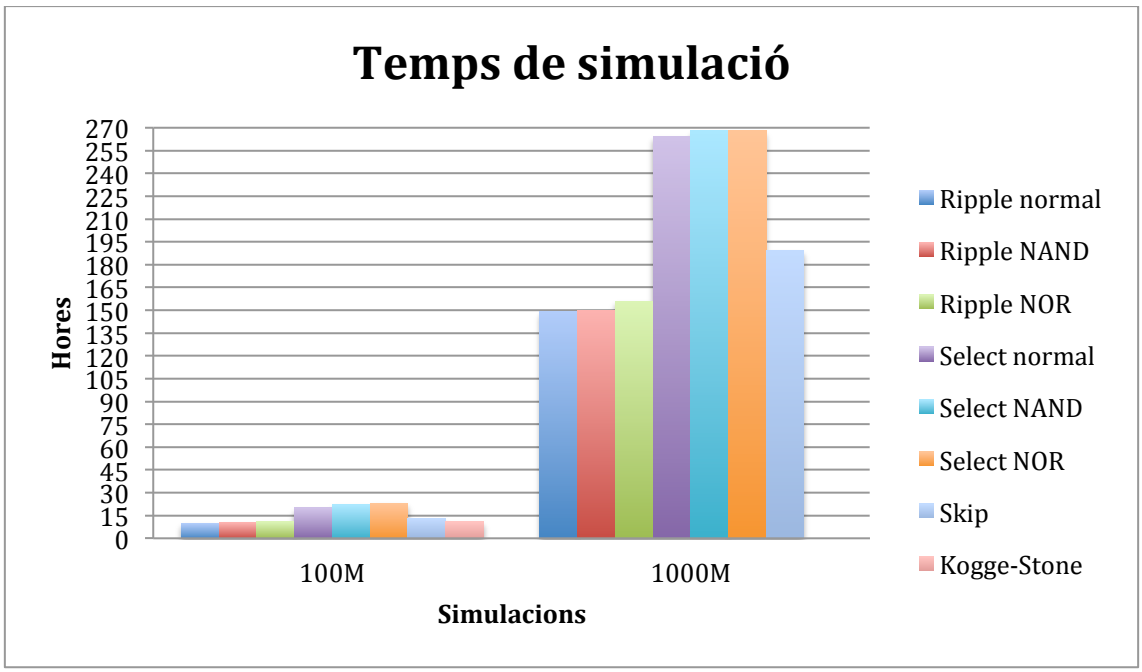

Figura 15: Augment del temps de 100M i 1000M de simulacions

Els gràfics superiors, podem veure l'impacte que té augmentar el nombre de simulacions per tal de precisar els resultats.
L'escala és força gran, passem de segons per cent mil simulacions, a minuts per un milió, una hora per deu milions, varies hores per cent milions i per últim dies per a mil milions.

En el gràfic s'han mostrat els vuit tipus de sumadors, tot i faltar la sèrie més llarga pel sumador Kogge-Stone. Es pot veure que com el comportament és similar en tots els casos i que a mesura que s'augmenta el nombre de simulacions per tal d'obtenir resultats més precisos, també ho fa el temps de forma molt ràpida.

11. Conclusions

Els resultats d'aquest projecte mostren que de les implementacions a nivell de bloc estudiades, la implementació *carry select* és la més robusta amb molta diferència davant errors transitoris, seguida de la Kogge-Stone, la *ripple carry* i per últim la *carry skip.*
Pel que fa a implementació per portes, la que demostra més robustesa fins i tot en redundància és la que utilitza portes *NOR*, seguida per la implementada amb portes *NAND*, i per últim la convencional.

Així doncs veiem que val la pena invertir, ja que la configuració més robusta de totes és el *carry select* amb portes *NOR*, això comporta un augment considerable de portes a nivell de *full adder* i el doble de *full adders* per la redundància. Ens trobem amb una configuració on la redundància a priori té l'objectiu d'agilitzar la propagació del *carry* i per tant fer el sumador més ràpid, però que a més converteix el sumador en una unitat amb percentatges d'emmascarament i tolerància a fallades molt importants.

El sumador *carry skip* està dissenyat també amb el propòsit d'agilitzar la propagació del *carry* i per tant augmentar la velocitat de les sortides. Resulta ser menys robust que totes les implementacions del sumador *ripple* amb molta diferència respecte al *ripple* implementat amb portes *NOR.* Això demostra que s'ha de tenir en compte en tot moment que moltes vegades millores en un aspecte d'una unitat com és el cas de la velocitat, poden afectar negativament altres aspectes importants com és el cas de la fiabilitat de la unitat.

Per altra part un bon exemple d'aprofitar molt bé l'augment de velocitat en la propagació i l'augment en portes és el sumador Kogge-Stone, que tot i els dos augments comentats manté la millor tolerància dels sumadors que no fan ús de la redundància.

12. Treball futur i continuïtat

Aquest projecte té un bon marge d'expansió. Hi ha dos possibles camins a seguir per tal d'ampliar el que s'ha treballat.

Una possibilitat de continuïtat és augmentar el volum d'unitats funcionals amb les quals treballar, queda algunes implementacions de sumadors per estudiar i a continuació es podria començar a treballar amb multiplicadors, divisors i les seves possibles implementacions.

Una altra extensió directa sigui a continuació d'aquest projecte o un cop estudiades la resta d'unitats funcionals, seria adaptar la implementació per a poder emular tots els circuits en una placa. Aquest procediment probablement s'hauria de desglossar en més d'un projecte, ja que adaptar els circuits per tal de ser capaços d'emular la injecció d'errors comporta treballar amb circuits bastant més extensos i per tant complexos.

Referencies

[1] Gil-Tomás, D., et al. "Studying the Effects of Intermittent Faults on a Microcontroller." *Microelectronics Reliability* 52.11 (2012): 2837-46. Print.

[2] Gracia, J., et al. "Analysis of the Influence of Intermittent Faults in a Microcontroller". *Proceedings - 2008 IEEE Workshop on Design and Diagnostics of Electronic Circuits and Systems, DDECS.* 2008. 80-85. Print.

[3] Navabi, Z. *Digital System Test and Testable Design: Using HDL Models and Architectures.*, 2011. Print.

[4] Dawson, Charles, Sathyam K. Pattanam, and David Roberts. "Verilog Procedural Interface for the Verilog Hardware Description Language". *Proceedings - IEEE International Verilog HDL Conference.* 1996. 17-23. Print.

[5] Pournaghdali, F., A. Rajabzadeh, and M. Ahmadi. "VHDLSFI: A Simulation-Based Multi-Bit Fault Injection for Dependability Analysis". *Proceedings of the 3rd International Conference on Computer and Knowledge Engineering, ICCKE 2013.* 2013. 354-360. Print.

[6] Boncalo, O., et al. "Cost Effective FPGA Probabilistic Fault Emulation". *NORCHIP 2014 - 32nd NORCHIP Conference: The Nordic Microelectronics Event.* 2015. Print.

[7] Na, J., and D. Lee. "Simulated Fault Injection using Simulator Modification Technique." *ETRI Journal* 33.1 (2011): 50-9. Print.